**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A1

(11) International Publication Number: WO 97/22069

(43) International Publication Date: 19 June 1997 (19.06.97)

(21) International Application Number: PCT/US96/19831

(22) International Filing Date: 10 December 1996 (10.12.96)

(30) Priority Data:
08/571,748 13 December 1995 (13.12.95) US

(71) Applicant: DIGITAL EQUIPMENT CORPORATION [US/US]; 111 Powdermill Road, Maynard, MA 01754 (US).

(72) Inventor: MONIER, Louis, M.; 2019 Maryland Street, Redwood City, CA 94061 (US).

(74) Agent: NATH, Rama, B.; Digital Equipment Corporation, 111 Powdermill Road, Maynard, MA 01754 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, GE, HU, JP, KE, KG, KP, KR, KZ, LK, LT, LU, LV, MD, MG, MN, MW, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, TJ, TT, UA, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

**Published**

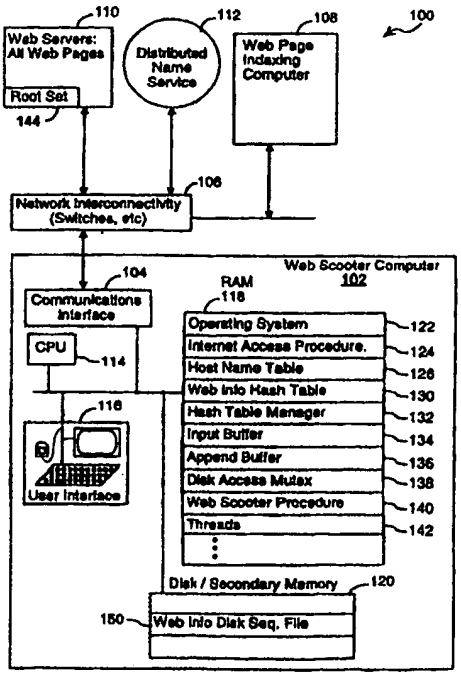
*With international search report.*

*Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.*

(54) Title: SYSTEM AND METHOD FOR LOCATING PAGES ON THE WORLD WIDE WEB AND FOR LOCATING DOCUMENTS FROM A NETWORK OF COMPUTERS

(57) Abstract

A Web crawler system and method for quickly fetching and analyzing Web pages on the World Wide Web or from computers connected by a network, includes a hash table stored in a random access memory (RAM) and a sequential Web information disk file. For every Web page known to the system, the Web crawler system stores an entry in the sequential disk file as well as a smaller entry in the hash table. The hash table entry includes a fingerprint value, a fetched flag that is set true only if the corresponding Web page has been successfully fetched, and a file location indicator that indicates where the corresponding entry is stored in the sequential disk file. Each sequential disk file entry includes the URL of a corresponding Web page, plus fetch status information concerning that Web page. All accesses to the Web information disk file are made sequentially via an input buffer such that a large number of entries from the sequential disk file are moved into the input buffer as single I/O operation. The sequential disk file is then accessed from the input buffer. Similarly, all new entries to be added to the sequential file are stored in an append buffer, and the contents of the append buffer are added to the end of the sequential whenever the append buffer is filled. In this way random access to the Web information disk file is eliminated, and latency caused by disk access limitations is minimized.



*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | |
|---|---|---|---|---|---|
| **AM** | Armenia | **GB** | United Kingdom | **MW** | Malawi |
| **AT** | Austria | **GE** | Georgia | **MX** | Mexico |
| **AU** | Australia | **GN** | Guinea | **NE** | Niger |
| **BB** | Barbados | **GR** | Greece | **NL** | Netherlands |
| **BE** | Belgium | **HU** | Hungary | **NO** | Norway |
| **BF** | Burkina Faso | **IE** | Ireland | **NZ** | New Zealand |
| **BG** | Bulgaria | **IT** | Italy | **PL** | Poland |
| **BJ** | Benin | **JP** | Japan | **PT** | Portugal |
| **BR** | Brazil | **KE** | Kenya | **RO** | Romania |
| **BY** | Belarus | **KG** | Kyrgystan | **RU** | Russian Federation |
| **CA** | Canada | **KP** | Democratic People's Republic of Korea | **SD** | Sudan |
| **CF** | Central African Republic | | | **SE** | Sweden |
| **CG** | Congo | **KR** | Republic of Korea | **SG** | Singapore |
| **CH** | Switzerland | **KZ** | Kazakhstan | **SI** | Slovenia |
| **CI** | Côte d'Ivoire | **LI** | Liechtenstein | **SK** | Slovakia |
| **CM** | Cameroon | **LK** | Sri Lanka | **SN** | Senegal |
| **CN** | China | **LR** | Liberia | **SZ** | Swaziland |
| **CS** | Czechoslovakia | **LT** | Lithuania | **TD** | Chad |
| **CZ** | Czech Republic | **LU** | Luxembourg | **TG** | Togo |
| **DE** | Germany | **LV** | Latvia | **TJ** | Tajikistan |
| **DK** | Denmark | **MC** | Monaco | **TT** | Trinidad and Tobago |
| **EE** | Estonia | **MD** | Republic of Moldova | **UA** | Ukraine |
| **ES** | Spain | **MG** | Madagascar | **UG** | Uganda |
| **FI** | Finland | **ML** | Mali | **US** | United States of America |
| **FR** | France | **MN** | Mongolia | **UZ** | Uzbekistan |
| **GA** | Gabon | **MR** | Mauritania | **VN** | Viet Nam |

# SYSTEM AND METHOD FOR LOCATING PAGES ON THE WORLD WIDE WEB AND FOR LOCATING DOCUMENTS FROM A NETWORK OF COMPUTERS

## FIELD OF THE INVENTION

The present invention relates generally to systems and method for accessing documents, called pages, on the World Wide Web (WWW), and for locating documents from a network of computers, and particularly to a system and method for quickly locating and analyzing pages on the World Wide Web.

## BACKGROUND OF THE INVENTION

Web documents, herein called Web pages, are stored on numerous server computers (hereinafter "servers") that are connected to the Internet. Each page on the Web has a distinct URL (universal resource locator). Many of the documents stored on Web servers are written in a standard document description language called HTML (hypertext markup language). Using HTML, a designer of Web documents can associate hypertext links or annotations with specific words or phrases in a document and specify visual aspects and the content of a Web page. The hypertext links identify the URLs of other Web documents or other parts of the same document providing information related to the words or phrases.

A user accesses documents stored on the WWW using a Web browser (a computer program designed to display HTML documents and communicate with Web servers) running on a Web client connected to the Internet. Typically, this is done by the user selecting a hypertext link (typically displayed by the Web browser as a highlighted word or phrase) within a document being viewed with the Web browser. The Web browser then issues a HTTP (hypertext transfer protocol) request for the requested document to the Web server identified by the requested document's URL. In response, the designated Web server returns the requested document to the Web browser, also using the HTTP.

As of the end of the 1995, the number of pages on the portion of the Internet known as the World Wide Web (hereinafter the "Web") had grown several fold during the prior one year period to at least 30 million pages. The present invention is directed at a system for keeping track of pages on the Web as the Web continues to grow.

The systems for locating pages on the Web are known variously as "Web crawlers," "Web spiders" and "Web robots." The present invention has been coined a "Web scooter" because it is so much faster than all known Web crawlers. The terms "Web crawler," "Web spider," "Web scooter," "Web crawler computer system," and "Web scooter computer system" are used interchangeably in this document.

Prior art Web crawlers work generally as follows. Starting with a root set of known Web pages, a disk file is created with a distinct entry for every known Web page. As additional Web pages are fetched and their links to other pages are analyzed, additional entries are made in the disk file to reference Web pages not previously known to the Web crawler. Each entry indicates whether or not the corresponding Web page has been processed as well as other status information. A Web crawler processes a Web page by (A) identifying all links to other Web pages in the page being processed and storing related information so that all of the identified Web pages that have not yet been processed are added to a list of Web pages to be processed or other equivalent data structure, and (B) passing the Web page to an indexer or other document processing system.

The information about the Web pages already processed is generally stored in a disk file, because the amount of information in the disk file is too large to be stored in random access memory (RAM). For example, if an average of 100 bytes of information are stored for each Web page entry, a data file representing 30 million Web pages would

occupy about 3 Gigabytes, which is too large for practical storage in RAM.

Next we consider the disk I/O incurred when processing one Web page. For purposes of this discussion we will assume that a typical Web page contains 20 references to other Web pages, and that a disk storage device can handle no more than 50 seeks per second. The Web crawler must evaluate each of the 20 page references in the page being processed to determine if it already knows about those pages. To do this it must attempt to retrieve 20 records from the Web information disk file. If the record for a specified page reference already exists, then that reference is discarded because no further processing is needed. However, if a record for a specified page is not found, an attempt must be made to locate a record for each possible alias of the page's address, thereby increasing the average of number of disk record seeks needed to analyze an average Web page to about 50 disk seeks per page.

If a disk file record for a specified page reference does not already exist a new record for the referenced page is created and added to the disk file, and that page reference is either added to a queue of pages to be processed, or the disk file entry is itself used to indicate that the page has not yet been fetched and processed.

Thus, processing a single Web page requires approximately 20 disk seeks (for reading existing records and for writing new records). As a result, given a limitation of 50 disk seeks per second, only about one Web page can be processed per second.

In addition, there is a matter of network access latency. On average, it takes about 3 seconds to retrieve a Web page, although the amount of time is highly variable depending on the location of the Web server and the particular hardware and software being used on both the Web

server and on the Web crawler computer. Network latency thus also tends to limit the number Web pages that can be processed by prior art Web crawlers to about 0.33 Web pages per second. Due to disk "seek" limitations, network latency, and other delay factors, a typical prior art Web crawler cannot process more than about 30,000 Web pages per day.

Due to the rate at which Web pages are being added to the Web, and the rate at which Web pages are being deleted and revised, processing 30,000 Web pages per day is inadequate for maintaining a truly current directory or index of all the Web pages on the Web. Ideally, a Web crawler should be able to visit (i.e., fetch and analyze) at least 2.5 million Web pages per day.

It is therefore desirable to have a Web crawler with such high speed capacity. An object of the present invention to provide an improved Web crawler that cam process millions of Web pages per day. It is a related goal of the present invention to provide an improved Web crawler that overcomes the aforementioned disk "seek" limitations and network latency limitations so as to enable the Web crawler's speed of operation to be limited primarily only by the processing speed of the Web crawler's CPU. It is yet another related goal of the present invention to provide a Web crawler system than can fetch and analyze, on average, at least 30 Web pages per second, and more preferably at least 100 Web pages per second.

## SUMMARY OF THE INVENTION

The invention, in its broad form, resides in a system for locating Web pages as recited in claim 1 and a method for locating Web pages as recited in claim 6.

Described hereinafter is a system and method for quickly locating and making a directory of Web pages on the World Wide Web. The Web crawler system includes a hash table stored in random access memory (RAM) and a sequential

file (herein called the "sequential disk file" or the "Web information disk file") stored in secondary memory, typically disk storage. For every Web page known to the system, the Web crawler system stores an entry in the sequential disk file as well as a smaller entry in the hash table. The hash table entry includes a fingerprint value, a fetched flag that is set true only if the corresponding Web page has been successfully fetched, and a file location indicator that indicates where the corresponding entry is stored in the sequential disk file. Each sequential disk file entry includes the URL of a corresponding Web page, plus fetch status information concerning that Web page.

All accesses to the Web information disk file are made sequentially via an input buffer such that a large number of entries from the sequential disk file are moved into the input buffer as single I/O operation. The sequential disk file is then accessed from the input buffer. Similarly, all new entries to be added to the sequential file are stored in an append buffer, and the contents of the append buffer are added to the end of the sequential disk file whenever the append buffer is filled. In this way random access to the Web information disk file is eliminated, and latency caused by disk access limitations is minimized.

The procedure for locating and processing Web pages includes sequentially reviewing all entries in the sequential file and selecting a next entry that meets with established selection criteria. When selecting the next file entry to process, the hash table is checked for all known aliases of the current entry candidate to determine if the Web page has already been fetched under an alias. If the Web page has been fetched under an alias, the error type field of the sequential file entry is marked as a "non-selected alias" and the candidate entry is not selected.

Once a next Web page reference entry has been selected, the Web crawler system attempts to fetch the

corresponding Web page. If the fetch is unsuccessful, the fetch status information in the sequential file entry for that Web page is marked as a fetch failure in accordance with the error return code returned to the Web crawler. If the fetch is successful, the fetch flag in the hash table entry for the Web page is set, as is a similar fetch flag in the sequential disk file entry (in the input buffer) for the Web page. In addition, each URL link in the fetched Web page is analyzed. If an entry for the URL referenced by the link or for any defined alias of the URL is already in the hash table, no further processing of the URL link is required. If no such entry is found in the hash table, the URL represents a "new" Web page not previously included in the Web crawler's database of Web pages and therefore an entry for the new Web page is added to the sequential disk file (i.e., it is added to the portion of the disk file in the append buffer). The new disk file entry includes the URL referenced by the link being processed, and is marked "not fetched". In addition, a corresponding new entry is added to the hash table, and the fetch flag of that entry is cleared to indicate that the corresponding Web page has not yet been fetched. In addition to processing all the URL links in the fetched page, the Web crawler sends the fetched page to an indexer for further processing.

**BRIEF DESCRIPTION OF THE DRAWINGS**

A more detailed understanding of the invention may be had from the following description of a preferred embodiment, given by way of example, and to be understood in conjunction with the accompanying drawings, wherein:

- Fig. 1 is a block diagram of a preferred embodiment of a Web crawler system in accordance with a preferred embodiment of the present invention.

- Fig. 2 is a block diagram of the hash table mechanism used in a preferred embodiment of the present invention.
- Fig. 3 is a block diagram of the sequential Web information disk file and associated data structures used in a preferred embodiment of the present invention.
- Fig. 4 is a flow chart of the Web crawler procedure used in a preferred embodiment of the present invention.

**DESCRIPTION OF THE PREFERRED EMBODIMENTS**

Referring to Fig. 1, there is shown a distributed computer system 100 having a Web scooter computer system 102. The Web scooter is connected by a communications interface 104 and a set of Internet and other network connections 106 to the Internet and a Web page indexing computer 108. In some embodiments the Web page indexing computer 108 is coupled directly to the Web scooter 102 through a private communication channel, without the use of a local or wide area network connection. The portions of the Internet to which the Web scooter 102 is connected are (A) Web servers 110 that store Web pages, and (B) servers that cooperate in a service known as the Distributed Name Service (DNS) collectively referenced here by reference numeral 112. For the purposes of this document it can be assumed that the DNS 112 provides any requester with the set of all defined aliases for any Internet host name, and that Internet host names and their aliases form a prefix portion of every URL.

In the preferred embodiment, the Web scooter 102 is an Alpha workstation computer made by Digital Equipment Corporation; however virtually any type of computer can be used as the Web scooter computer. In the preferred embodiment, the Web scooter 102 includes a CPU 114, the

previously mentioned communications interface 104, a user interface 116, random access memory (RAM) 118 and disk memory (disk) 120. In the preferred embodiment the communications interface 104 is a very high capacity communications interface that can handle 1000 or more overlapping communication requests with an average fetch throughput of at least 30 Web pages per second.

In the preferred embodiment, the Web scooter's RAM has a Gigabyte of random access memory and stores:

- a multitasking operating system 122;
- an Internet communications manager program 124 for fetching Web pages as well as for fetching alias information from the DNS 112;
- a host name table 126, which stores information representing defined aliases for host names;
- a Web information hash table 130;
- a hash table manager procedure 132;
- an input buffer 134 and an append buffer 136;
- a mutex 138 for controlling access to the hash table 130, input buffer 134 and append buffer 136;
- a Web scooter procedure 140; and
- thread data structures 142 for defining T1 threads of execution, where the value of T1 is an integer selectable by the operator of the Web scooter computer system 102 (e.g., T1 is set at a value of 1000 in the preferred embodiment).

Disk storage 120 stores a Web information disk file 150 that is sequentially accessed through the input buffer 134 and append buffer 136, as described in more detail below.

The host name table 126 stores information representing, among other things, all the aliases of each host name that are known to the DNS 112. The aliases are effectively a set of URL prefixes which are substituted by the Web scooter procedure 140 for the host name portion of

a specified Web page's URL to form a set of alias URLs for the specified Web page.

The use and operation of the above mentioned data structures and procedures will next be described with reference to Figs. 1 through 4 and with reference to Tables 1 and 2. Tables 1 and 2 together contain a pseudocode representation of the Web scooter procedure. While the pseudocode employed here has been invented solely for the purposes of this description, it utilizes universal computer language conventions and is designed to be easily understandable by any computer programmer skilled in the art.

**Web Information Hash Table**

Referring to Fig. 2, the Web information hash table 130 includes a distinct entry 160 for each Web page that has been fetched and analyzed by the Web scooter system as well as each Web page referenced by a URL link in a Web page that has been fetched and analyzed. Each such entry includes:

- a fingerprint value 162 that is unique to the corresponding Web page;
- a one bit "fetched flag" 164 that indicates whether or not the corresponding Web page has been fetched and analyzed by the Web scooter; and
- a file location value 166 that indicates the location of a corresponding entry in the Web information disk file 150.

In the preferred embodiment, each fingerprint value is 63-bits long, and the file location values are each 32-bits long. As a result each hash table entry 160 occupies exactly 12 bytes in the preferred embodiment. While the exact size of the hash table entries is not important, it is important that each hash table entry 160 is

significantly smaller (e.g., at least 75% smaller on average) than the corresponding disk file entry.

The hash table manager 132 receives, via its "interface" 170, two types of procedure calls from the Web scooter procedure 140:

- a first request asks the hash table manager 132 whether or not an entry exists for a specified URL, and if so, whether or not the fetched flag of that record indicates that the corresponding Web page has previously been fetched and analyzed; and
- a second request asks the hash table manager to store a new entry in the hash table 130 for a specified URL and a specified disk file location.

The hash table manager 132 utilizes a fingerprint hash function 172 to compute a 63-bit fingerprint for every URL presented to it. The fingerprint function 172 is designed to ensure that every unique URL is mapped into a similarly unique fingerprint value. The fingerprint function generates a compressed encoding of any specified Web page's URL. The design of appropriate fingerprint functions is understood by persons of ordinary skill in the art. It is note that while there are about $2^{25}$ to $2^{26}$ Web pages, the fingerprints can have $2^{63}$ distinct values.

When the Web scooter procedure 140 asks the hash table manager 132 whether or not the hash table already has an entry for a specified URL, the hash table manager (A) generates a fingerprint of the specified URL using the aforementioned fingerprint hash function 172, (B) passes that value to a hash table position function 174 that determines where in the hash table 130 an entry having that fingerprint value would be stored, (C) determines if such an entry is in fact stored in the hash table, (D) returns a failure value (e.g., -1) if a matching entry is not found, and (E) returns a success value (e.g., 0) and fetched flag

value and disk position value of the entry if the entry is found in the hash table.

In the preferred embodiment, the hash table position function 174 determines the position of a hash table entry based on a predefined number of low order bits of the fingerprint, and then follows a chain of blocks of entries for all fingerprints with the same low order bits. Entries 160 in the hash table 130 for a given value of the low order bits are allocated in blocks of B1 entries per block, where B1 is a tunable parameter. The above described scheme used in the preferred embodiment has the advantage of storing data in a highly dense manner in the hash table 130. As will be understood by those skilled in the art, many other hash table position functions could be used.

When the Web scooter procedure 140 asks the hash table manager 132 to store a new hash table entry for a specified URL and a specified disk file location, the hash table manager (A) generates a fingerprint of the specified URL using the aforementioned fingerprint hash function 172, (B) passes that value to a hash table position function 174 that determines where in the hash table 130 an entry having that fingerprint value should be stored, and (C) stores a new entry 160 in the hash table at the determined position, with a fetch flag value that indicates the corresponding Web page has not yet been fetched, and also containing the fingerprint value and the specified disk file position.

**Web Information Disk File and Buffers**

Referring to Fig. 3 and Table 2, disk access operations are minimized through the use of an input buffer 134 and an append buffer 136, both of which are located in RAM. Management of the input and append buffers is performed by a background sequential disk file and buffer handler procedure, also known as the disk file manager.

In the preferred embodiment, the input buffer and append buffer are each 50 to 100 Megabytes in size. The input buffer 134 is used to store a sequentially ordered contiguous portion of the Web information disk file 150. The Web scooter procedure maintains a pointer 176 to the next entry in the input buffer to be processed, a pointer 178 to the next entry 180 in the Web information disk file 150 to be transferred to the input buffer 134, as well as a number of other bookkeeping pointers required for coordinating the use of the input buffer 134, append buffer 136 and disk file 150.

All accesses to the Web information disk file 150 are made sequentially via the input buffer 134 such that a large number of entries from the sequential disk file are moved into the input buffer as single I/O operation. The sequential disk file 150 is then accessed from the input buffer. Similarly, all new entries to be added to the sequential file are stored in the append buffer 136, and the contents of the append buffer are added to the end of the sequential whenever the append buffer is filled. In this way random access to the Web information disk file is eliminated, and latency caused by disk access limitations is minimized.

Each time all the entries in the input buffer 134 have been scanned by the Web scooter, all updates to the entries in the input buffer are stored back into the Web information disk file 150 and all entries in the append buffer 136 are appended to the end of the disk file 150. In addition, the append buffer 136 is cleared and the next

set of entries in the disk file, starting immediately after the last set of entries to be copied into the input buffer 134 (as indicated by pointer 178), are copied into the input buffer 134. When the last of the entries in the disk file have been scanned by the Web scooter procedure, scanning resumes at the beginning of the disk file 150.

Whenever the append buffer 136 is filled with new entries, its contents are appended to the end of the disk file 150 and then the append buffer is cleared to receive new entries.

Each entry 180 in the Web information disk file 150 stores:

- a variable length URL field 182 that stores the URL for the Web page referenced by the entry;
- a fetched flag 184 that indicates whether or not the corresponding Web page has been fetched and analyzed by the Web scooter;
- a timestamp 186 indicating the date and time the referenced Web page was fetched, analyzed and indexed;
- a size value 188 indicating the size of the Web page;
- an error type value 190 that indicates the type of error encountered, if any, the last time an attempt was made to fetch the referenced Web page or if the entry represents a duplicate (i.e., alias URL) entry that should be ignored; and
- other fetch status parameters 192 not relevant here.

Because the URL field 182 is variable in length, the records 180 in the Web information disk file 150 are also variable in length.

**Web Scooter Procedure**

Referring now to Figs. 1-4 and the pseudocode in Table 1, the Web scooter procedure 140 in the preferred embodiment works as follows. When the Web scooter

procedure begins execution, it initializes (200) the system's data structures by:

- scanning through a pre-existing Web information disk file 150 and initializing the hash table 130 with entries for all entries in the sequential disk file;
- copying a first batch of sequential disk entries from the disk file 150 into the input buffer 134;
- defining an empty append buffer 136 for new sequential file entries; and
- defining a mutex 138 for controlling access to the input buffer 134, append buffer 136 and hash table 130.

The Web scooter initializer then launches T1 threads (e.g., 1000 threads are launched in the preferred embodiment), each of which executes the same scooter procedure.

The set of entries in the pre-existing Web information disk file 150, prior to execution of the Web scooter initializer procedure, is called the "root set" 144 of known Web pages. The set of "accessible" Web pages consists of all Web pages referenced by URL links in the root set and all Web pages referenced by URL links in other accessible Web pages. Thus it is possible that some Web pages are not accessible to the Web scooter 102 because there are no URL link connections between the root set and those "inaccessible" Web pages.

When information about such Web pages becomes available via various channels, the Web information disk file 150 can be expanded (thereby expanding the root set 144) by "manual" insertion of additional entries or other mechanisms to include additional entries so as to make accessible the previously inaccessible Web pages.

The following is a description of the Web scooter procedure executed by all the simultaneously running threads. The first step of the procedure is to request and

wait for the mutex (202). Ownership of the mutex is required so that no two threads will process the same disk file entry, and so that no two threads attempt to write information at the same time to the hash table, input buffer, append buffer or disk file. The hash table 130, input buffer 134, append buffer 136 and disk file 150 are herein collectively called the "protected data structures," because they are collectively protected by use of the mutex. Once a thread owns the mutex, it scans the disk file entries in the input buffer, beginning at the next entry that has not yet been scanned (as indicated by pointer 176), until is locates and selects an entry that meets defined selection criteria (204).

For example, the default selection criteria is: any entry that references a Web page denoted by the entry as never having been fetched, or which was last fetched and analyzed more than H1 hours ago, where H1 is a operator selectable value, but excluding entries whose error type field indicates the entry is a duplicate entry (i.e., a "non-selected alias," as explained below). If H1 is set to 168, all entries referencing Web pages last fetched analyzed more than a week ago meet the selection criteria. Another example of a selection criteria, in which Web page size is taken into account, is: an entry representing a Web page that has never been fetched, or a Web page of size greater than S1 that was last fetched and analyzed more than H1 hours ago, or a Web page or size S1 or less that was last fetched and analyzer more than H2 hours ago, but excluding entries whose error type field indicates the entry is a "non-selected alias," where S1, H1 and H2 are operator selectable values.

When selecting the next entry to process, the hash table is checked for all known aliases of the current entry candidate to determine if the Web page has already been fetched under an alias. In particular, if an entry meets the defined selection criteria, all known aliases of the

URL for the entry are generated using the information in the host name table 126, and then the hash table 130 is checked to see if it stores an entry for any of the alias URLs with a fetched flag that indicates the referenced Web page has been fetched under that alias URL. If the Web page referenced by the current entry candidate in the input buffer is determined to have already been fetched under an alias URL, the error type field 190 of that input buffer entry is modified to indicate that this entry is a "non-selected alias," which prevents the entry from being selected for further processing both at this time and in the future.

Once a Web page reference entry has been selected, the mutex is released so that other threads can access the protected data structures (206). Then the Web scooter procedure attempts to fetch the corresponding Web page (208). After the fetch completes or fails the procedure once again requests and waits for the mutex (210) so that it can once again utilize the protected data structures.

If the fetch is unsuccessful (212-N), the fetch status information in the sequential file entry for that Web page is marked as a fetch failure in accordance with the error return code returned to the Web crawler (214). If the fetch is successful (212-Y), the fetch flag 164 in the hash table entry 160 for the Web page is set, as is the fetch flag 184 in the sequential disk file entry 180 (in the input buffer) for the Web page. In addition, each URL link in the fetched Web page is analyzed (216).

After the fetched Web page has been analyzed, or the fetch failure has been noted in the input buffer entry, the mutex is released so that other threads can access the protected data structures (218) .

The procedure for analyzing the URL links in the fetched Web page is described next with reference to Fig. 4B. It is noted here that a Web page can include URL links to documents, such as image files, that do not contain

information suitable for indexing by the indexing system 108. These referenced documents are often used as components of the Web page that references them. For the purposes of this document, the URL links to component files such as image files and other non-indexable files are not "URL links to other Web pages." These URL links to non-indexable files are ignored by the Web scooter procedure.

Once all the URL links to other Web pages have been processed (230), the fetched Web page is sent to the indexer for indexing (232) and the processing of the fetched Web page by the Web scooter is completed. Otherwise, a next URL link to a Web page is selected (234). If there is already a hash table entry for the URL associated with the selected link (236), no further processing of that link is required and a next URL link is selected (234) if there remain any unprocessed URL links in the Web page being analyzed.

If there isn't already a hash table entry for the URL associated with the selected link (236), all known aliases of the URL for the entry are generated using the information in the host name table 126, and then the hash table 130 is checked to see if it stores an entry for any of the alias URLs (238). If there is an entry in the hash table for any of the alias URLs, no further processing of that link is required and a next URL link is selected (234) if there remain any unprocessed URL links in the Web page being analyzed.

If no entry is found in the hash table for the selected link's URL or any of its aliases, the URL represents a "new" Web page not previously included in the Web crawler's database of Web pages and therefore an entry for the new Web page is added to the portion of the disk file in the append buffer (240). The new disk file entry includes the URL referenced by the link being processed, and is marked "not fetched". In addition, a corresponding

new entry is added to the hash table, and the fetch flag of that entry is cleared to indicate that the corresponding Web page has not yet been fetched (240). Then processing of the Web page continues with the next unprocessed URL link in the Web page (234), if there remain any unprocessed URL links in the Web page.

The Web information hash table 130 is used, by procedures whose purpose and operation are outside the scope of this document, as an index into the Web information disk file 150 because the hash table 130 includes disk file location values for each known Web page. In other words, an entry in the Web information disk file is accessed by first reading the disk file address in the corresponding entry in the Web information hash table and then reading the Web information disk file entry at that address.

**Alternative Embodiments**

Any data structure that has the same properties of the Web information hash table 130, such as a balanced tree, a skip list, or the like, could be used in place of the hash table structure 130 of the preferred embodiment.

As a solution, the present invention uses three primary mechanisms to overcome the speed limitations of prior art Web crawlers.

First, a Web page directory table is stored in RAM with sufficient information to determine which Web pages links represent new Web pages not previously known to the Web crawler, enabling received Web pages to be analyzed without having to access a disk file.

Second, a more complete Web page directory is accessed only in sequential order, and performing those accesses via large input and append buffers that reduce the number of disk accesses performed to the point that disk accesses do not have a significant impact on the speed performance of the Web crawler.

Third, by using a large number of simultaneously active threads to execute the Web scooter procedure, and by providing a communications interface capable of handling a similar number of simultaneous communication channels to Web servers, the present invention avoids the delays caused by network access latency.

In particular, while numerous ones of the threads are waiting for responses to Web page fetch requests, other ones of the threads are analyzing received Web pages. By using a large number of threads all performing the same Web scooter procedure, there will tend to be, on average, a queue of threads with received Web pages that are waiting for the mutex so that they can process the received Web pages. Also, the Web page fetches will tend to be staggered over time. As a result, the Web scooter is rarely in a state where it is waiting to receive a Web page and has no other work to do. Throughput of the Web scooter can then be further increased by using a multiprocessor workstation and further increasing the number of threads that are simultaneously executing the Web scooter procedure.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may be made without departing from the scope of the invention as presented and claimed herein.

**TABLE 1**

Pseudocode Representation of Web Scooter Procedure

```
Procedure: Web Scooter
{
/* Initialization Steps   */
Scan through pre-existing Web information disk file and
   initialize Hash Table with entries for all entries in
   the sequential file
Read first batch of sequential disk entries into input
   buffer in RAM
Define empty Append Buffer for new sequential file entries
Define Mutex for controlling access to Input Buffer, Append
   Buffer and Hash Table
Launch 1000 Threads, each executing same Scooter Procedure
}

Procedure: Scooter
{
Do Forever:
   {
   Request and Wait for Mutex
   Read sequential file (in Input Buffer) until a new URL
      to process is selected in accordance with established
      URL selection criteria.  When selecting next URL to
      process, check Hash Table for all known aliases of URL
      to determine if the Web page has already been fetched
      under an alias, and if the Web page has been fetched
      under an alias mark the Error Type field of the
      sequential file entry as a "non-selected alias."
      /* Example of Selection Criteria: URL has never been
            fetched, or was last fetched more than H1
            hours ago, and is not a non-selected alias   */
   Release Mutex
```

```
Fetch selected Web page
Request and Wait for Mutex
If fetch is successful
   {
   Mark page as fetched in Hash Table and Sequential File
      entry in Input Buffer
   /* Analyze Fetched Page   */
   For each URL link in the page
      {
      If URL or any defined alias is already in the Hash
         Table
         {    Do Nothing     }
      Else
         {
         /* the URL represents a "New" Web Page not
            previously included in the database   */
         Add new entry for corresponding Web page to the
            Append Buffer, with entry marked "not fetched"
         Add entry to Hash Table, with entry marked "not
            fetched"
         }
      }
      Send Fetched Page to Indexer for processing
   }
Else
   {
   Mark the entry in Input Buffer currently being
    processed with appropriate "fetch failure" error
    indicator based on return code received
   }
Release Mutex
}    /* End of Do Forever Loop    */
}
```

**TABLE 2**

Pseudocode Representation for Background Sequential File Buffer Handler

```
Procedure: Background Sequential File Buffer Handler (a/k/a
   the disk file manager)
{
Whenever a "read sequential file" instruction overflows the
   Input Buffer
   {
   Copy the Input Buffer back to the sequential disk file
   Read next set of entries into Input Buffer
   Append contents of Append Buffer to the end of the
      sequential disk file
   Clear Append Buffer to prepare for new entries
   }

Whenever an "add entry to sequential file" causes the
   Append Buffer to Overflow
   {
   Append contents of Append Buffer to the end of the
      sequential disk file
   Clear Append Buffer to prepare for new entries
   Add pending new entry to the beginning of the Append
      Buffer
   }
}
```

**WHAT IS CLAIMED IS:**

1. A system for locating data sets comprising Web pages stored on remotely located accessible computers, each Web page having a unique URL (universal resource locator), at least some of said Web pages including URL links to other ones of the Web pages, the system comprising:

a communications interface for fetching specified ones of the Web pages from said remotely located computers in accordance with corresponding URLs;

a Web information file having a set of entries, each entry denoting, for a corresponding Web page, a URL and fetch status information;

a Web information table, stored in RAM (random access memory), having a set of entries, each entry denoting a fingerprint value and fetch status information for a corresponding Web page; and

means to implement a Web scooter procedure, executed by the system, for fetching and analyzing Web pages, said Web scooter procedure including instructions for fetching Web pages whose Web information file entries meet predefined selection criteria based on said fetch status information, for determining for each URL link in each received Web page whether a corresponding entry already exists in the Web information table, and for each URL link which does not have a corresponding entry in the Web information table adding a new entry in the Web information table and a corresponding new entry in the Web information file.

2. The system of claim 1, including multiple threads that each execute the Web scooter procedure during overlapping time periods, including means such that while some of the threads are fetching Web pages, other ones of the Web pages are analyzing fetched Web pages.

3. The system of claim 2, including a mutex, wherein said Web scooter procedure executed by each of the threads includes instructions for requesting and waiting for the mutex before accessing the Web information table and Web information file.

4. The system of claim 3, including:

an input buffer and an append buffer;

a file manager for storing blocks of sequentially ordered entries from the Web information file into the input buffer;

said Web scooter procedure scanning and analyzing Web information file entries in the input buffer to locate said Web information file entries that meet said predefined selection criteria;

said Web scooter procedure storing in said append buffer all entries to be added to Web information said file; and

said file manager for moving multiple entries in the append buffer to the Web information file.

5. The system of claim 1, wherein each of the entries in the second memory include an address of a corresponding entry in the first memory.

6. A method of locating data sets comprising Web pages stored on remotely located but accessible computers, each Web page having a unique URL (universal resource locator), at least some of said Web pages including URL links to other ones of the Web pages, comprising the steps of:

storing a Web information file having a set of entries, each entry denoting, for a corresponding Web page, a URL and fetch status information;

storing in RAM (random access memory) a Web information table having a set of entries, each entry denoting a fingerprint value and fetch status information for a corresponding Web page; and

executing a Web scooter procedure, for fetching and analyzing Web pages, including (A) sequentially scanning entries in the Web information file do determine which of said entries meet predefined selection criteria, (B) fetching Web pages whose Web information file entries meet said predefined selection criteria, (C) determining for each URL link to another Web page in each received Web page whether a corresponding entry already exists in the Web information table, and (D) for each URL link which does not have a corresponding entry in the Web information table adding a new entry in the Web information table and a corresponding new entry in the Web information file.

7. The method of claim 6, including executing said Web scooter procedure in multiple threads during overlapping time periods, such that while some of the threads are fetching Web pages, other ones of the Web pages are analyzing fetched Web pages.

8. The method of claim 7, including:

defining a mutex;

while executing said Web scooter procedure in each of said threads, requesting and waiting for the mutex before accessing the Web information table and Web information file.

9. The method of claim 8, including the steps of:

defining an "input buffer" and an "append buffer" in said RAM;

storing blocks of sequentially ordered entries from the Web information file into the input buffer;

said step of sequentially scanning entries in the Web information file comprising the step of including scanning the Web information file entries in the input

buffer to determine which of said Web information file entries meet said predefined selection criteria;

storing in said append buffer all entries to be added to said file; and

moving multiple entries in the append buffer to the Web information file.

10. The method of claim 6, wherein each of the entries in the Web information table includes an address of a corresponding entry in the Web information file, said method including:

accessing one of said entries in said Web information file by reading the address in a corresponding ones of the entries in the Web information table and then reading said one entry in said Web information file at said address.

11. An apparatus for locating data sets stored on computers connected by a network, each data set being uniquely identified by an address, at least some of the data sets including one or more linked addresses of other data sets stored on the computers, comprising:

a communications interface connected to the network for sending requests to the computers for identified ones of the data sets, each request including the address of the identified one of the data sets, and for receiving data sets in response to said requests;

a first memory storing a first set of entries, each entry of the first set including the address of a corresponding data set and status information for the corresponding data set;

a second memory storing a second set of entries, each entry of the second set including an encoding of the address of a corresponding data set and an

encoding of status information for the corresponding data set; and

thread means, coupled to the first and second memories and to the communications interface, for sequentially reading the entries of the first set, generating the requests for those identified ones of the data sets that have corresponding entries in the first set that meet predefined status-based selection criteria, and, in response to receiving the identified data sets, creating new entries in said first and second sets corresponding to each of at least a subset of the addresses in the received data sets for which there is no corresponding entry in the second set.

12. The apparatus of claim 11, wherein each of the entries in the second set include an address of a corresponding entry in the first set, said second set of entries being for indexing the first set of entries.

13. The apparatus of claim 11, including a multiplicity of said thread means such that while some of the thread means are generating said requests and receiving said identified data sets, other ones of the thread means are creating new entries in said first and second memories.

14. The apparatus of claim 13, including a mutex, wherein each of said thread means includes logic for requesting and waiting for the mutex before accessing the first memory and second memory.

15. The apparatus of claim 14, includes:

an input buffer and an append buffer, located in said second memory;

a manager that stores in the input buffer sequentially ordered groups of the entries in the first memory;

each of said thread means including means for scanning and analyzing entries in the input buffer to locate said entries that meet said predefined status-based selection criteria; and

each of said thread means storing in said append buffer all entries to be added to said first memory;

said manager also having means for moving multiple entries in the append buffer to the first memory.

16. A method of locating data sets stored on computers connected by a network, each data set being uniquely identified by an address, at least some of said data sets including one or more linked addresses of other data sets stored on the computers, comprising the steps of:

(A) storing in a first memory a first set of entries, each entry in said first set including the address of a corresponding data set and status information for the corresponding data set;

(B) storing in a second memory a second set of entries, each entry in said second set including an encoding of the address of a corresponding data set and an encoding of status information for the corresponding data set;

(C) sequentially reading the entries of the first set;

(D) transmitting requests via the network to the computers for those identified ones of the data sets that have corresponding entries of the first set that meet predefined status-based selection criteria; and

(E) creating, in response to receiving the identified ones of the data sets, new entries in said first and second sets corresponding to each of at least a subset of the addresses in the received data sets for which there is no corresponding entry in the second set.

17. The method of claim 16, wherein said step B includes storing in each of the entries in the second set an address of a corresponding entry in the first set, said second set of entries being for indexing the first set of entries.

18. The method of claim 16, including performing steps C, D and E in multiple threads during overlapping time periods, such that while some of the threads are fetching data sets, other ones of the data sets are analyzing fetched data sets.

19. The method of claim 18, including:

defining a mutex; and

each of said threads requesting and waiting for the mutex before accessing the first and second sets of entries in the first and second memories.

20. The method of claim 19, including:

defining an input buffer and an append buffer in said second memory;

storing blocks of sequentially ordered entries from the first set of entries into the input buffer;

said sequentially reading step comprising the step of sequentially reading the entries in the input buffer and determining which of said input buffer entries meet said predefined status-based selection criteria;

storing in said append buffer all entries to be added to said first memory; and

moving multiple entries in the append buffer to the first memory.

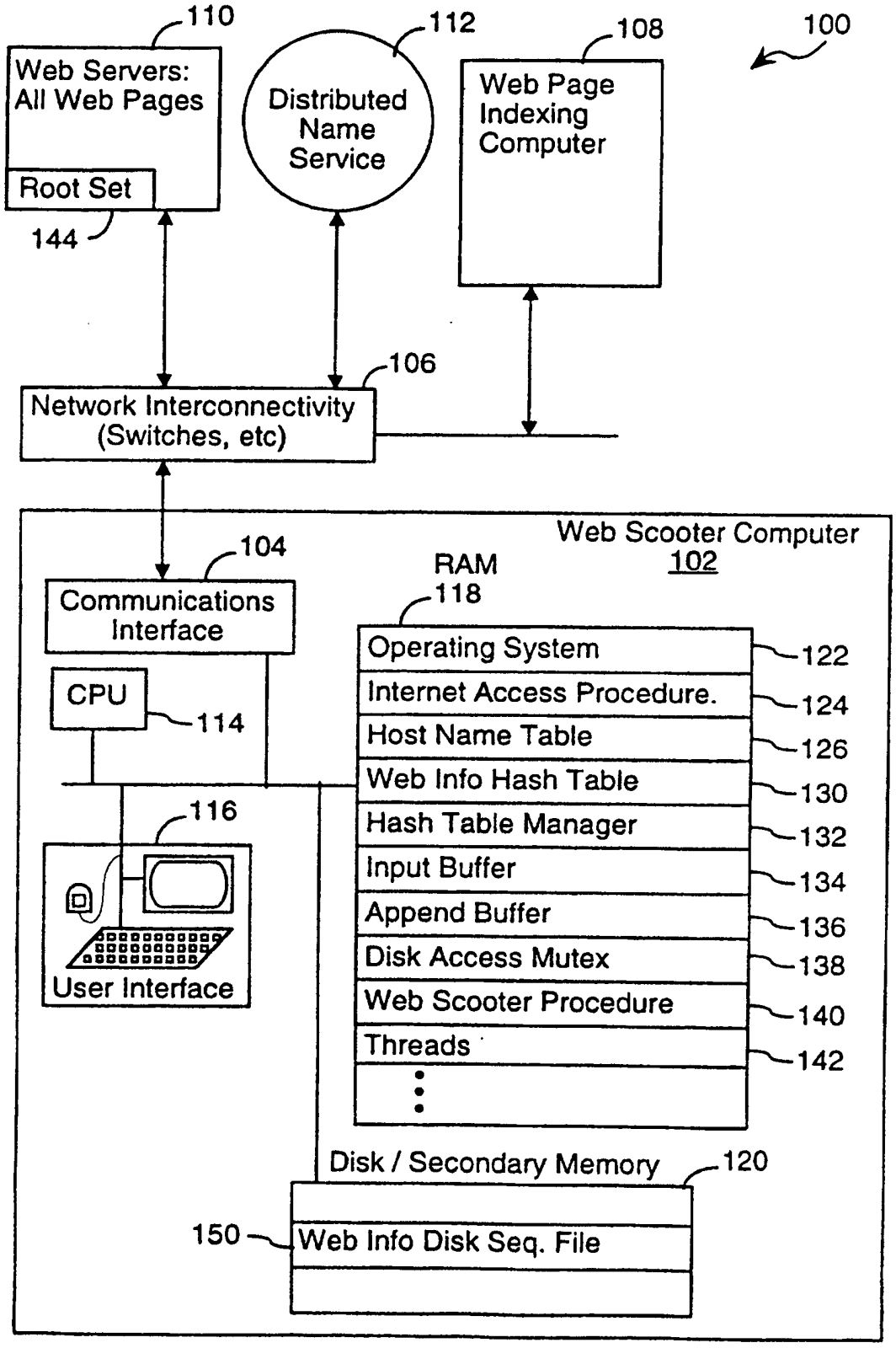

110
Web Servers:
All Web Pages
Root Set
144
112
Distributed
Name
Service
108
Web Page
Indexing
Computer
100
106
Network Interconnectivity
(Switches, etc)
Web Scooter Computer
102
104
Communications
Interface
CPU
114
RAM
118
Operating System
122
Internet Access Procedure.
124
Host Name Table
126
Web Info Hash Table
130
Hash Table Manager
132
Input Buffer
134
Append Buffer
136
Disk Access Mutex
138
Web Scooter Procedure
140
Threads
142
116
User Interface
Disk / Secondary Memory
120
150
Web Info Disk Seq. File

FIG. 1

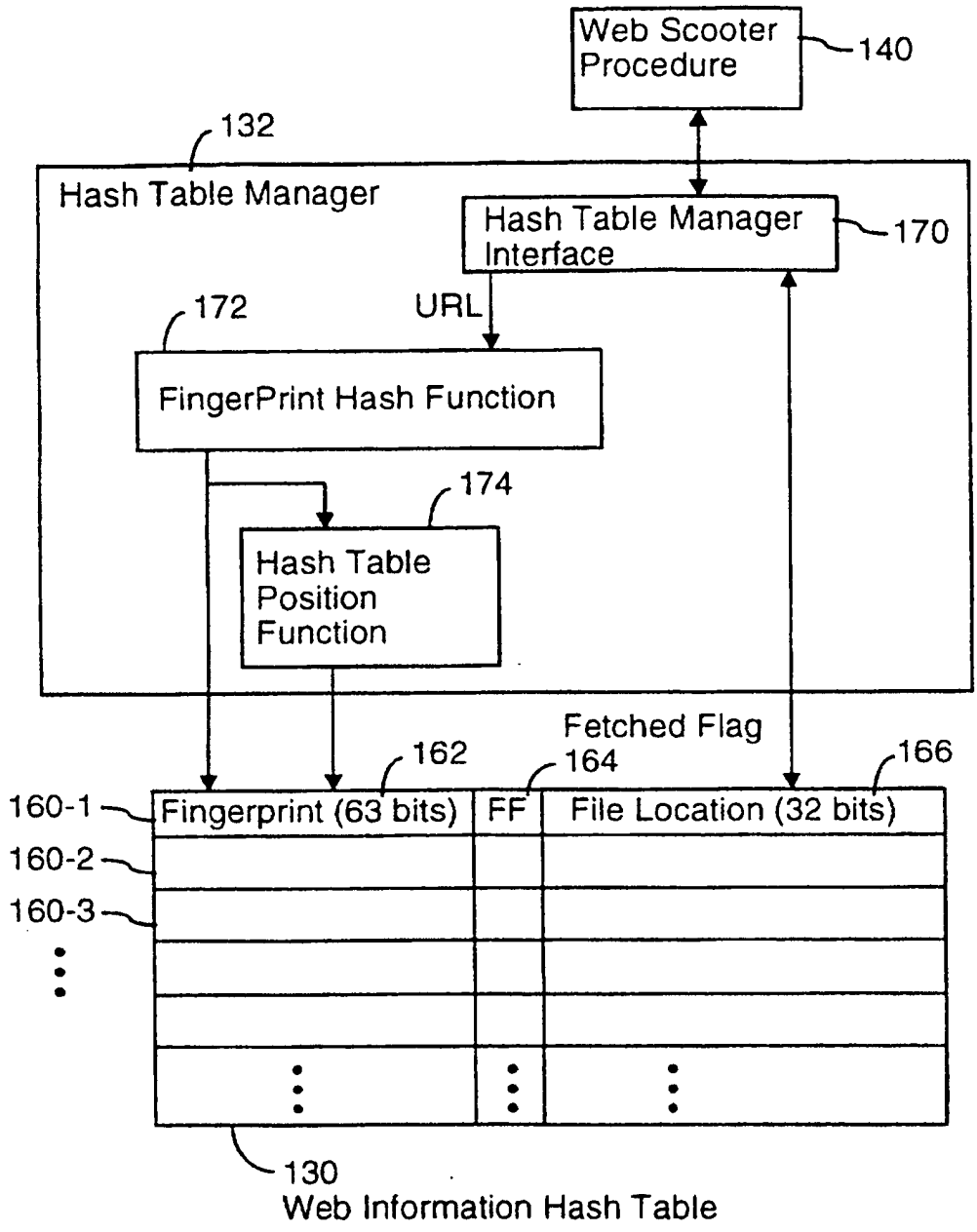

Web Scooter Procedure
140
132
Hash Table Manager
Hash Table Manager Interface
170
172
URL
FingerPrint Hash Function
174
Hash Table Position Function
Fetched Flag
162
164
166
160-1
160-2
160-3
Fingerprint (63 bits)
FF
File Location (32 bits)
130
Web Information Hash Table

FIG. 2

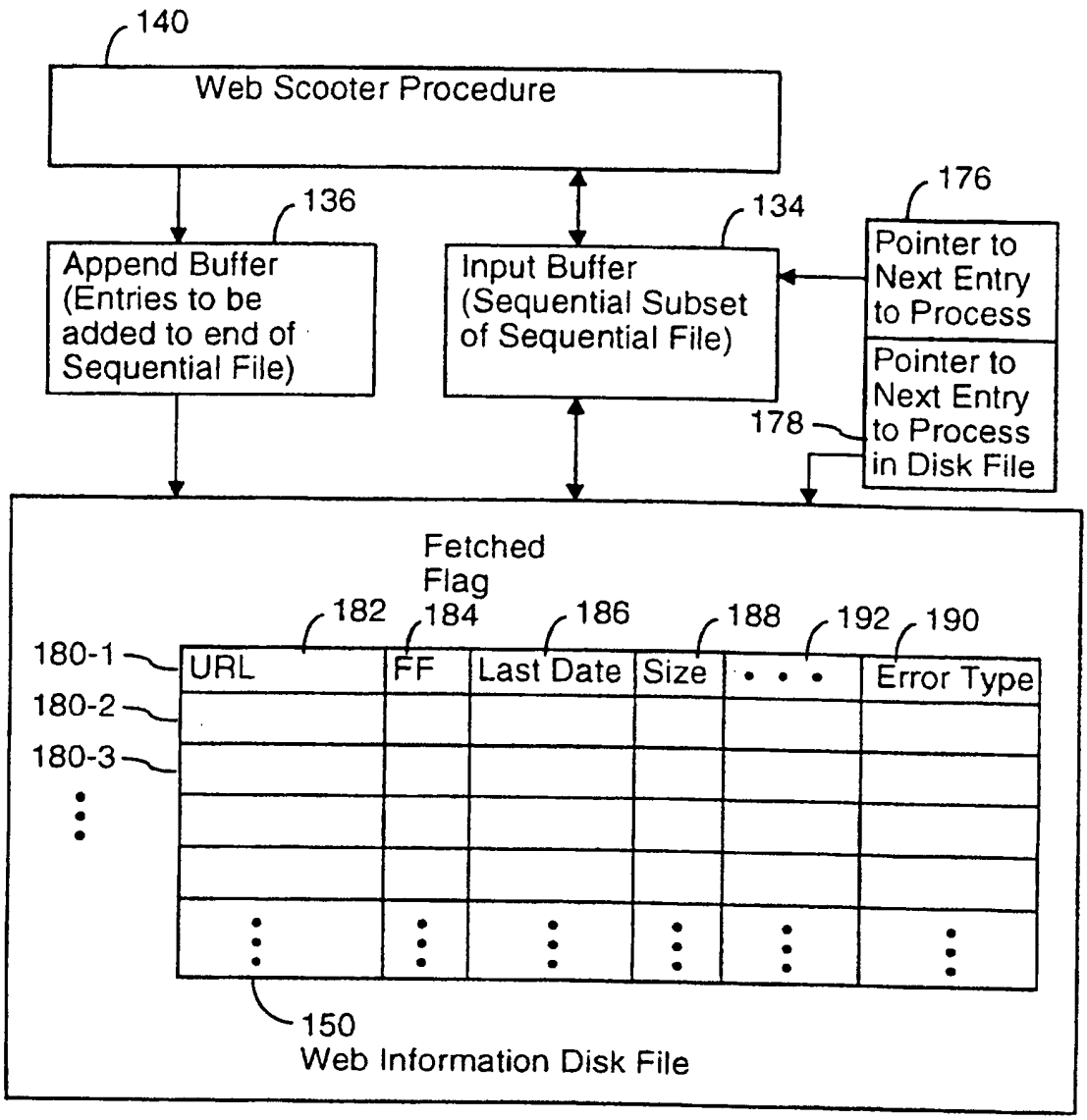

140
Web Scooter Procedure
136
Append Buffer (Entries to be added to end of Sequential File)
134
Input Buffer (Sequential Subset of Sequential File)
176
Pointer to Next Entry to Process
Pointer to Next Entry to Process in Disk File
178
Fetched Flag
182
184
186
188
192
190
180-1
180-2
180-3
URL
FF
Last Date
Size
Error Type
150
Web Information Disk File

FIG. 3

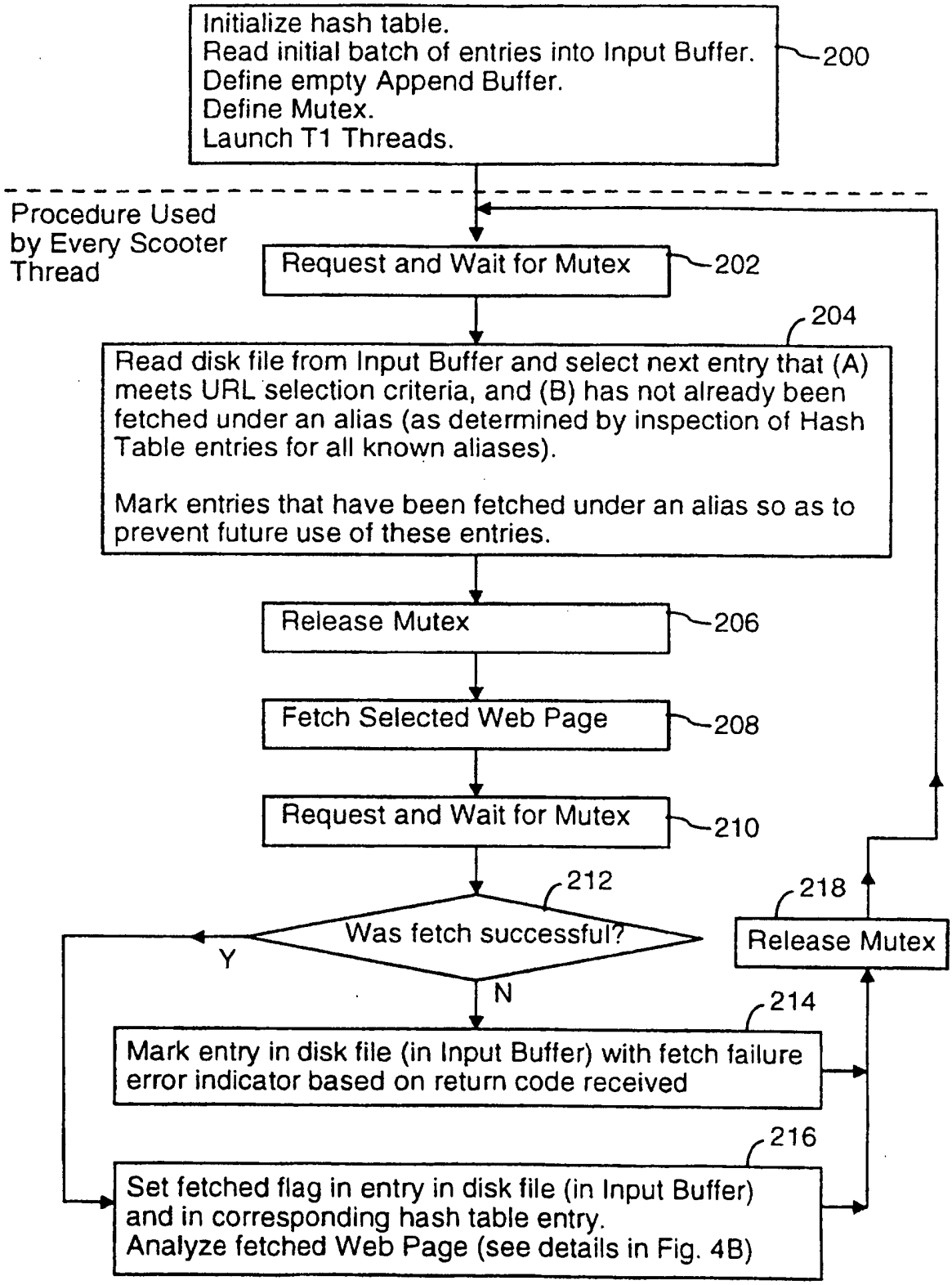

Initialize hash table.
Read initial batch of entries into Input Buffer.
Define empty Append Buffer.
Define Mutex.
Launch T1 Threads.
200
Procedure Used by Every Scooter Thread
Request and Wait for Mutex
202
204
Read disk file from Input Buffer and select next entry that (A) meets URL selection criteria, and (B) has not already been fetched under an alias (as determined by inspection of Hash Table entries for all known aliases).
Mark entries that have been fetched under an alias so as to prevent future use of these entries.
Release Mutex
206
Fetch Selected Web Page
208
Request and Wait for Mutex
210
212
Was fetch successful?
Y
N
218
Release Mutex
214
Mark entry in disk file (in Input Buffer) with fetch failure error indicator based on return code received
216
Set fetched flag in entry in disk file (in Input Buffer) and in corresponding hash table entry.
Analyze fetched Web Page (see details in Fig. 4B)

FIG. 4A

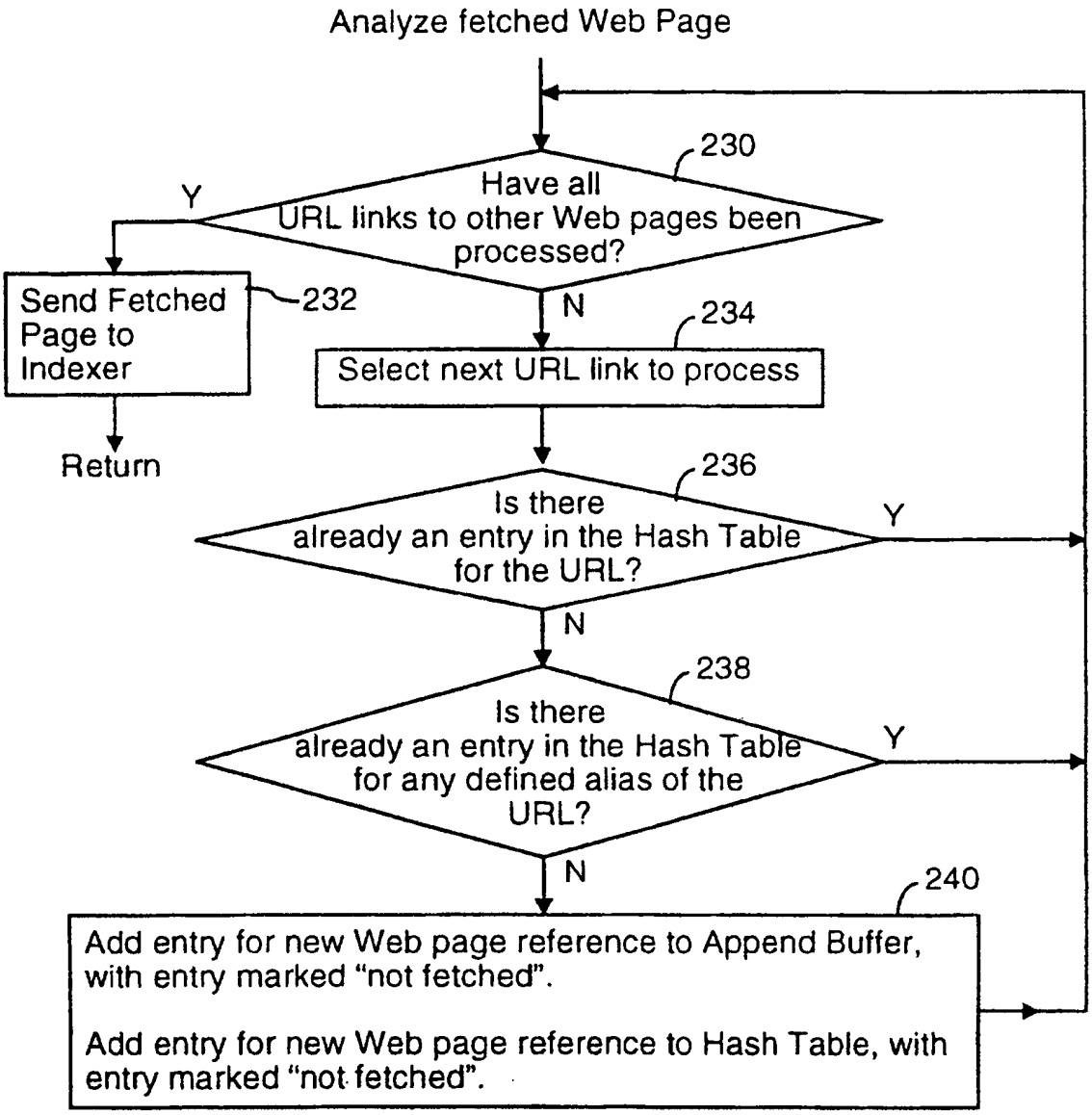



FIG. 4B

| FIG. 4A |
| --- |
| FIG. 4B |

FIG. 4

# INTERNATIONAL SEARCH REPORT

Inter nal Application No
PCT/US 96/19831

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

| Category * | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | PERSONAL COMPUTER MAGAZINE, JAN. 1996, VNU BUSINESS PUBLICATIONS, UK, pages 90-92, 94, 97 - 98, 100, XP000646762 SIMPSON D ET AL: "The searchers World Wide Web search engines" see the whole document --- | 1,6,11, 16 |
| A | ELECTRONIC LIBRARY, OCT. 1995, LEARNED INFORMATION, UK, vol. 13, no. 5, ISSN 0264-0473, pages 467-476, XP000647883 SHA V T: "Cataloguing Internet resources: the library approach" see the whole document ----- | 1,6,11, 16 |

☐ Further documents are listed in the continuation of box C.

☐ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

| Date of the actual completion of the international search | Date of mailing of the international search report |
|---|---|
| 4 April 1997 | 16.04.97 |
| Name and mailing address of the ISA<br>European Patent Office, P.B. 5818 Patentlaan 2<br>NL - 2280 HV Rijswijk<br>Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,<br>Fax (+31-70) 340-3016 | Authorized officer<br>Katerbau, R |

Form PCT/ISA/210 (second sheet) (July 1992)